Hamlet R-package: step-by-step user instructions

Teemu Daniel Laajala

March 1, 2016

Contents

1 Introduction

`Hamlet` is an R package intended for the statistical analysis of pre-clinical studies. This document is a basic introduction to the functionality of `hamlet` and a general overview to the analysis workflow of preclinical studies.

This document is structured as follows: First, a general overview to inputting and processing the raw data is presented. Second, functionality is presented for the processing of pre-intervention data. Finally, functionality is presented for the post-intervention period, along with brief discussion on the differences between non-matched and matched statistical approaches. Each section comes with a list of useful functions specific for the subtask.

Latest version of `hamlet` is available in the Comprehensive R Archive Network (CRAN, http://cran.r-project.org/). CRAN mirrors are by default available in the installation of R, and the `hamlet` package is installable using the R terminal command: `install.packages("hamlet")`. This should prompt the user to select a nearby CRAN mirror, after which the installation of `hamlet` is automatically performed. After the `install.packages`-call, the `hamlet` package can be loaded with either command `library("hamlet")` or `require("hamlet")`.

The following notation is used in the document: R commands, package names and function names are written in `typewriter font`. The notation of format `pckgName::funcName` indicates that the function `funcName` is called from the package `pckgName`. If only the function name is given, this indicates that it is located in the base package in R and is thus always available.

1.1 Analysis workflow

Two different types of case-control setups for the analysis of pre-clinical are presented in Fig. 1.

The type A experiment design in Fig. 1 is preferred, as matching is performed before allocation to the experiment groups, and therefore improves the balance and power of the experiment. The alternate experiment type B requires the bipartite matching task, where suitable pairs of individuals are identified over two or more groups that existed prior to matching. This document focuses on experiment design of type A, where similarity information is utilized readily before interventions.

2 Pre-intervention analyses

2.1 Loading data into R

The `hamlet` package comes pre-installed with the VCaP dataset, which is used here to illustrate the workflow. Two different formats of the data are provided. First one is available in `data(vcapwide)`, which includes the data in the so-called *wide* format. In this data format the columns are indicators for different variables available for the experimental unit (here animal). For example, the two first rows of observations are extracted with:

```
> require(hamlet)
> data(vcapwide)
```

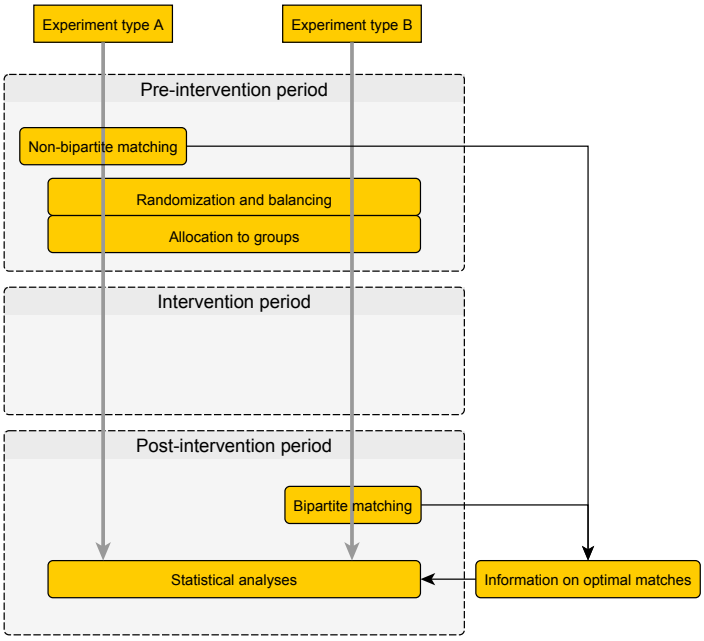

Figure 1: Analysis workflow for pre-clinical experiments

```
> vcapwide[1:2,]
```

```
      CastrationDate CageAtAllocation   Group TreatmentInitiationWeek    Submatch
ID003         100413            13489 Vehicle                 Week10  Submatch_1
ID007         170413            13810     MDV                 Week10 Submatch_10
         ID PSAWeek2 PSAWeek3 PSAWeek4 PSAWeek5 PSAWeek6 PSAWeek7 PSAWeek8 PSAWeek9
ID003 ID003     7.67    14.76    24.78     2.03     5.97     8.16    13.72    16.57
ID007 ID007     2.01     5.17     8.59    14.62     1.99     2.81     4.23     5.38
      PSAWeek10 PSAWeek11 PSAWeek12 PSAWeek13 PSAWeek14 BWWeek0 BWWeek1 BWWeek2
ID003     21.30     45.69     54.50     53.55     27.64    30.5    31.7    32.6
ID007      7.55      9.70     17.45     22.79     21.88    28.8    30.0    30.6
      BWWeek3 BWWeek4 BWWeek5 BWWeek6 BWWeek7 BWWeek8 BWWeek9 BWWeek10 BWWeek11
ID003    33.8    33.9    32.2    32.6    32.6    33.2    34.2     35.0     36.1
ID007    31.6    32.9    32.4    32.0    31.1    30.3    30.5     31.6     31.7
      BWWeek12 BWWeek13 BWWeek14
ID003     37.9     37.5     39.7
ID007     32.4     33.5     33.3
```

An another format of the same dataset is provided in `data(vcaplong)`. This is the data from the same experiment in the so-called *long* format, where only few column variables are available (here PSA or body weight), and the different observations belonging to a single experimental unit (here animal) are distinguished using the measurement time (variable *Week* or *DrugWeek*). Again, first few rows of the dataset:

```
> data(vcaplong)
> vcaplong[1:3,]
```

	A	B	C	D
1	Animal	PSA week 10 [ug/l]	PSA week 9 [ug/l]	Body weight week 10 [g]
2	ID003	21,3	16,57	35
3	ID007	7,55	5,38	31,6
4	ID008	23,58	17,4	33,6
5	ID009	13,17	11,14	31,7
6	ID010	9,9	9,33	34,1
7	ID016	15,05	15,29	39,6
8	ID018	13,53	12,14	34
9	ID025	13,13	10,91	33,3
10	ID027	9,59	8,79	32
11	ID031	7,04	6,95	36,6
12	ID032	8,49	8,02	34,9
13	ID037	13,74	13,38	32,4
14	ID040	23,62	19,15	35,9
15	ID045	14,27	9,8	34,8
16	ID047	6,57	6,28	31,9
17	ID054	34,72	27,14	32,1
18	ID056	28,15	22,05	32,2
19	ID058	9,74	7,68	34

Figure 2: Example Excel-format data, where rows correspond to individuals and columns to different characteristics at baseline. The single sheet data can be easily exported in a text-based format such as CSV.

```
     PSA  log2PSA   BW   Submatch    ID Week DrugWeek   Group Vehicle ARN MDV
11 21.30 4.412782 35.0 Submatch_1 ID003   10        0 Vehicle       1   0   0
12 45.69 5.513807 36.1 Submatch_1 ID003   11        1 Vehicle       1   0   0
13 54.50 5.768184 37.9 Submatch_1 ID003   12        2 Vehicle       1   0   0
```

The former *wide* format is useful for summarizing multiple variables when constructing distance matrices for the data. The latter *long* format is typically used for longitudinal mixed-effects modeling where observations are correlated through time.

2.2 Excel format data

An example view of a pre-clinical dataset is given in Fig. 2. Such a dataset can be saved in an R-friendly format by selecting option `File > Save As` and `CSV (Comma delimited)` as the save format in MS Excel.

2.2.1 CSV-files

CSV (Comma Delimited Values) is a suitable text-based format for the data to be read into R using either the function `read.table` or `read.csv`. The above presented example CSV file can be opened with the following command:

```
> ex <- read.table(file="example.csv", sep=";", dec=",", stringsAsFactors=F, header=T)
> ex

   Animal PSA.week.10..ug.l. PSA.week.9..ug.l. Body.weight.week.10..g.
1   ID003              21.30             16.57                    35.0
```

2	ID007	7.55	5.38	31.6
3	ID008	23.58	17.40	33.6
4	ID009	13.17	11.14	31.7
5	ID010	9.90	9.33	34.1
6	ID016	15.05	15.29	39.6
7	ID018	13.53	12.14	34.0
8	ID025	13.13	10.91	33.3
9	ID027	9.59	8.79	32.0
10	ID031	7.04	6.95	36.6
11	ID032	8.49	8.02	34.9
12	ID037	13.74	13.38	32.4
13	ID040	23.62	19.15	35.9
14	ID045	14.27	9.80	34.8
15	ID047	6.57	6.28	31.9
16	ID054	34.72	27.14	32.1
17	ID056	28.15	22.05	32.2
18	ID058	9.74	7.68	34.0

The above presented CSV file was read into R using `read.table` with the following parameters: `file="example.csv"` is the first parameter and indicates the input file from our current working directory. The working directory may be changed using the command `setwd` or by including its path in the file parameter, i.e. `file="D://my//current//windows//working//directory//example.csv"`. `sep=";"` indicates that the values on each line are separated with the symbol ';', as is the format defined for the CSV delimited files with ","-decimals. This could also be a value such as `\tab` or `" "` (space). `dec=","` indicates that the "," symbol is used for decimals. The default value for indicating decimals is "." otherwise. `stringsAsFactors=F` indicates that strings should not be handled as factors. Factors are an R class, where a character string may only take instances of a predetermined set of strings. As each of our animal IDs - which are read as strings - are unique, it is generally more flexible to conserve them as character strings. Lastly, `header=T` indicates that the text CSV file has a header row as the first row, which includes names for each column. If this value is set to `header=F` or `header=FALSE`, the first row of the text file is read as the first observation and the columns are left unnamed.

Depending on the country of origin, the CSV files may use "." decimals and "," separator, or alternatively (as assumed here) "," decimals" and ";" separators.

List of useful functions:

- `read.table`, `read.csv`
- `data`: `data(vcaplong)`, `data(vcapwide)`

2.3 Distance and dissimilarity functions

A distance or dissimilarity function is used to describe the amount of dissimilarity between two experimental units. Common choices for computing the amount of similarity between two vectors $\mathbf{x}$ and $\mathbf{y}$ include:

- Euclidean distance: $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$.

- Standardized Euclidean distance: $\sqrt{\sum_{i=1}^{P} \frac{(x_i - y_i)^2}{s_i^2}}$
- Mahalanobis distance: $\sqrt{(x-y)^T S^{-1}(x-y)}$

Here, $\mathbf{x}$ and $\mathbf{y}$ are expected to be observation vectors of length P, where each dimension describes the measured value for a particular covariate. S describes the covariance-variance matrix between covariates, and therefore incorporates inter-correlations between variables. The standard deviation s may be used to standardize differences in variation over the dimensions.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0.00	18.05	2.80	10.32	13.53	7.87	9.00	10.08	14.38	17.28	15.40	8.61	3.58	9.76	18.23	17.33	9.21	14.62
2	18.05	0.00	20.14	8.05	5.23	14.78	9.34	8.04	3.99	5.27	4.33	10.15	21.60	8.66	1.36	34.81	26.51	3.98
3	2.80	20.14	0.00	12.29	15.89	10.64	11.35	12.30	16.50	19.79	17.82	10.70	2.89	12.08	20.39	14.87	6.67	16.92
4	10.32	8.05	12.29	0.00	4.44	9.12	2.53	1.62	4.29	8.90	6.47	2.42	13.82	3.55	8.20	26.84	18.54	5.39
5	13.53	5.23	15.89	4.44	0.00	9.61	4.59	3.68	2.19	4.48	2.08	5.83	16.97	4.45	5.02	30.61	22.33	1.66
6	7.87	14.78	10.64	9.12	9.61	0.00	6.60	7.91	11.39	11.95	10.86	7.56	10.10	7.33	14.57	24.16	16.49	10.84
7	9.00	9.34	11.35	2.53	4.59	6.60	0.00	1.47	5.54	8.71	6.57	2.04	12.43	2.58	9.34	26.03	17.75	5.85
8	10.08	8.04	12.30	1.62	3.68	7.91	1.47	0.00	4.33	7.98	5.70	2.70	13.59	2.19	8.15	27.04	18.73	4.73
9	14.38	3.99	16.50	4.29	2.19	11.39	5.54	4.33	0.00	5.57	3.20	6.20	17.87	5.55	3.93	31.12	22.81	2.29
10	17.28	5.27	19.79	8.90	4.48	11.95	8.71	7.98	5.57	0.00	2.48	10.19	20.60	7.98	4.77	34.56	26.32	3.82
11	15.40	4.33	17.82	6.47	2.08	10.86	6.57	5.70	3.20	2.48	0.00	7.91	18.81	6.05	3.96	32.58	24.30	1.58
12	8.61	10.15	10.70	2.42	5.83	7.56	2.04	2.70	6.20	10.19	7.91	0.00	11.96	4.34	10.10	25.09	16.82	7.14
13	3.58	21.60	2.89	13.82	16.97	10.10	12.43	13.59	17.87	20.60	18.81	11.96	0.00	13.27	21.73	14.19	6.53	18.11
14	9.76	8.66	12.08	3.55	4.45	7.33	2.58	2.19	5.55	7.98	6.05	4.34	13.27	0.00	8.95	26.95	18.69	5.07
15	18.23	1.36	20.39	8.20	5.02	14.57	9.34	8.15	3.93	4.77	3.96	10.10	21.73	8.95	0.00	35.04	26.73	4.05
16	17.33	34.81	14.87	26.84	30.61	24.16	26.03	27.04	31.12	34.56	32.58	25.09	14.19	26.95	35.04	0.00	8.31	31.72
17	9.21	26.51	6.67	18.54	22.33	16.49	17.75	18.73	22.81	26.32	24.30	16.82	6.53	18.69	26.73	8.31	0.00	23.42
18	14.62	3.98	16.92	5.39	1.66	10.84	5.85	4.73	2.29	3.82	1.58	7.14	18.11	5.07	4.05	31.72	23.42	0.00

Table 1: Euclidean distance matrix D for 18 animals

Table 1 shows the Euclidean distance matrix for the 18 animals presented in Figure 2.

List of useful functions:

- `dist` includes many common distance and dissimilarity functions (Euclidean by default, others: `method="manhattan"`, `method="maximum"`, `method="minkowski"`
- `cluster::daisy`, daisy includes Gower's dissimilarity for mixed data (parameter `metric="gower"`)

2.4 Non-bipartite optimal matching of animals at baseline (BB)

The non-bipartite optimal matching problem may be solved using the provided branch and bound algorithm:

```
> sol.bb <- match.bb(d, g=3)

[1] "Performing initial sorting for a good initial guess"
[1] "Computing boundaries for minimum distances in possible combinations..."
[1] "Starting branch and bound"
[1] "Branches: 272"
[1] "Bounds: 7140"
[1] "Ends visited: 25"
```

```
[1] "Solution cost 169.62"
[1] "Solution: 5,3,5,6,4,5,6,4,3,2,2,6,1,4,3,1,1,2"
```

```
> submatches <- paste("Submatch_", LETTERS[1:6][sol.bb$solution], sep="")
> names(submatches) <- names(sol.bb$solution)
> submatches
```

```
          1           2           3           4           5           6
"Submatch_E" "Submatch_C" "Submatch_E" "Submatch_F" "Submatch_D" "Submatch_E"
          7           8           9          10          11          12
"Submatch_F" "Submatch_D" "Submatch_C" "Submatch_B" "Submatch_B" "Submatch_F"
         13          14          15          16          17          18
"Submatch_A" "Submatch_D" "Submatch_C" "Submatch_A" "Submatch_A" "Submatch_B"
```

The `match.bb` function returns the solution to the optimal matching task. It takes as input a distance matrix `d`, as is indicated in the function call `match.bb(d, g=3)` (notice that `d` was defined before). Furthermore, the size of the submatches is defined using the parameter `g=3`. This value indicates that the optimal matching algorithm minimizes edges within triplets. Each observation has to belong to a triplet called a submatch.

List of useful functions:

- Multigroup non-bipartite matching: `hamlet::match.bb`
- Paired non-bipartite matching: `hamlet::match.bb`, `nbpMatching::nonbimatch`
- Paired bipartite matching: `optmatch::fullmatch`

2.5 Non-bipartite optimal matching of animals at baseline (GA)

While the above described Branch and Bound algorithm is guaranteed to identify the global optimum, in some cases it is not feasible due to size of the search tree. In such cases, a feasible optimum is easily detected using a Genetic Algorithm implemention provided in `hamlet::match.ga`.

The Genetic Algorithm (GA) commonly includes many parameters, as it aims to mimic evolutionary processes in solving a problem, here a non-bipartite multigroup matching problem. The basic parameters that should be considered include `generations`, which indicates for how many generations the simulation is run for and thus increases run time approximately linearly, and the parameter `popsize`, which indicates how many solutions should be "living" inside the whole population at a given generation. A thumb rule is that many solutions are easily solvable by the 1,000th generation, if the population size is at least 100, but the user may want to use the visualizations and diagnostic plots to see how well the GA has managed to solve the optimization problem. The convergence happens over the generations similarly as presented in Figure 3.

```
> sol.bb[["cost"]] # Guaranteed global optimum
```

```
[1] 169.62
```

```
> sol.ga[[3]] # Identified solution by GA
```

```
> set.seed(1) # GA is a stochastic algorithm, fixing the seed for reproducibility
> sol.ga <- match.ga(d, g=3, generations=100, popsize=100)

[1] "Best found solution vector:"
 [1] 5 6 5 3 1 5 1 3 6 2 2 3 4 1 6 4 4 2
[1] "Best found solution cost:"
[1] 171.72
```

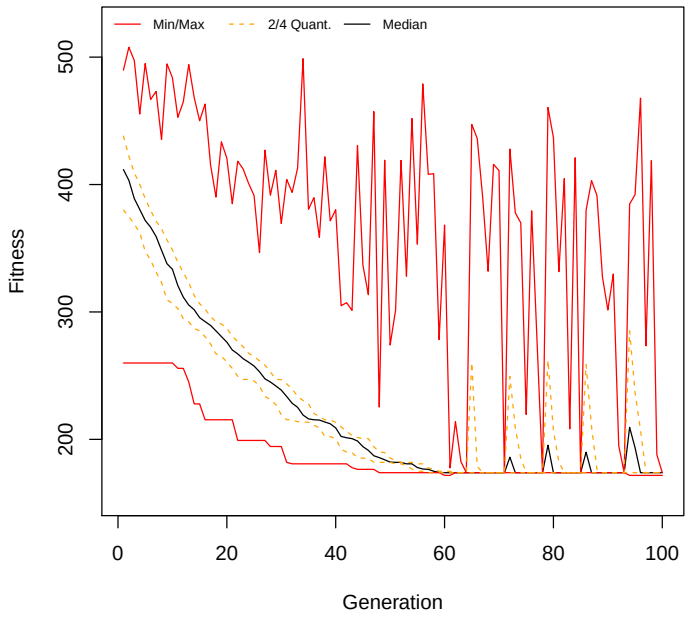

Figure 3: Convergence of the GA in the given optimization problem. The minimum shows the best identified optimization solution, while the quantiles give insight to the solution heterogeneity living in the solution space.

```
[1] 171.72
```

The GA algorithm, with a linear run time, has resulted in a very close optimum to the guaranteed global optimum identified using BB, which may in return have some cases lead to drastic increases in run times. Both algorithms may thus be applicable where appropriate.

2.6 Randomization based on matched individuals

The submatches identified in the above section should not be mistaken for the randomly allocated intervention groups. The final intervention groups are obtained by dividing members of each submatch in the found solution to a separate treatment arm. Since the within-submatch distances are minimized, this guarantees that comparable individuals are randomly divided to separate arms:

```
> ex[,"Submatch"] <- submatches
> set.seed(1) # for reproducibility
> ex[,"AllocatedGroups"] <- match.allocate(ex[,"Submatch"])
> ex <- ex[order(ex[,"Submatch"]),] # Sort for submatches
```

	Animal	PSA.week.10..ug.l.	PSA.week.9..ug.l.	Body.weight.week.10..g.	Submatch	AllocatedGroups
13	ID040	23.62	19.15	35.90	Submatch_A	Group_B
16	ID054	34.72	27.14	32.10	Submatch_A	Group_C
17	ID056	28.15	22.05	32.20	Submatch_A	Group_A
10	ID031	7.04	6.95	36.60	Submatch_B	Group_C
11	ID032	8.49	8.02	34.90	Submatch_B	Group_A
18	ID058	9.74	7.68	34.00	Submatch_B	Group_B
2	ID007	7.55	5.38	31.60	Submatch_C	Group_C
9	ID027	9.59	8.79	32.00	Submatch_C	Group_A
15	ID047	6.57	6.28	31.90	Submatch_C	Group_B
5	ID010	9.90	9.33	34.10	Submatch_D	Group_A
8	ID025	13.13	10.91	33.30	Submatch_D	Group_C
14	ID045	14.27	9.80	34.80	Submatch_D	Group_B
1	ID003	21.30	16.57	35.00	Submatch_E	Group_A
3	ID008	23.58	17.40	33.60	Submatch_E	Group_C
6	ID016	15.05	15.29	39.60	Submatch_E	Group_B
4	ID009	13.17	11.14	31.70	Submatch_F	Group_C
7	ID018	13.53	12.14	34.00	Submatch_F	Group_B
12	ID037	13.74	13.38	32.40	Submatch_F	Group_A

Table 2: The result table in variable `ex` after performing the optimal matching and allocation.

As is seen Table 2, each submatch (column `Submatch`) consists of similar experimental units in terms of the baseline characteristics (i.e. PSA and body weight). Furthermore, the baseline data has now been allocated in such a manner, that each submatch evenly distributes to the proposed intervention groups (column `AllocatedGroup`), resulting in balanced baseline intervention groups. These artificial labels `A`, `B`, and `C` may then be given to an external experimenter in a blinded manner, and allocated to the true labels in any fashion without any pre-fixed control group, as all pairwise contrasts have been considered in the submatching procedure.

List of useful functions:

- Multigroup non-bipartite matching: `hamlet::match.bb`, `hamlet::match.ga`

```
> boxplot(PSA.week.10..ug.l. ~ AllocatedGroups, data = ex, range=0,
+ xlab="Group", ylab="PSA week 10 ul/g")
```

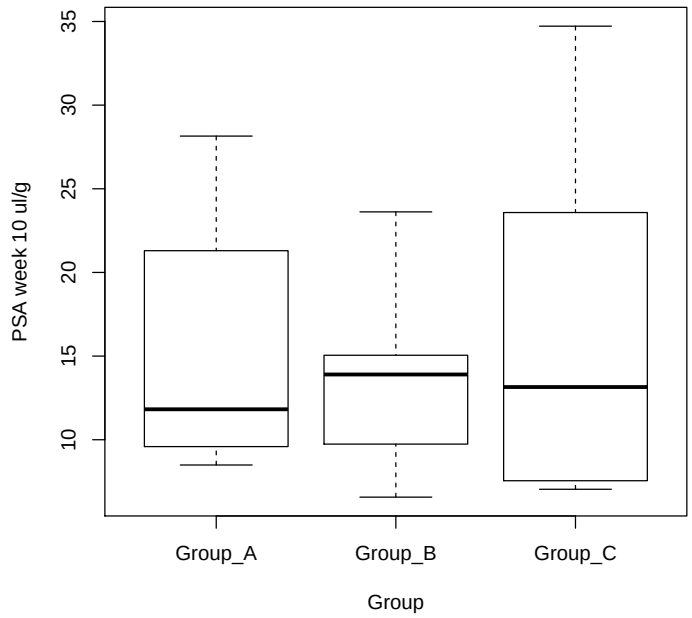

Figure 4: Boxplots for the week 10 PSA in the example allocation

- Paired non-bipartite matching: `hamlet::match.bb`, `hamlet::match.ga`, `nbpMatching::nonbimatch`
- Paired bipartite matching: `optmatch::fullmatch`

2.7 Visualizations for pre-clinical data

Various visualization functions are available to illustrate baseline balance. For example, the boxplots in respect to allocation groups can be plotted using a command such as `boxplot`, which is illustrated in Figure 4.

Mixed variable scatterplots with annotations for the submatches or allocation groups are plotted using the function `hamlet::mixplot`, which can be seen in Figures 5 or 6 respectively.

List of useful functions:

- Scatterplots etc: `hamlet::mixplot`, `plot`, `boxplot`
- Heatmaps: `hamlet::hmap`, `heatmap`, `gplots::heatmap.2`

```
> mixplot(ex[,2:5], pch=16)
```

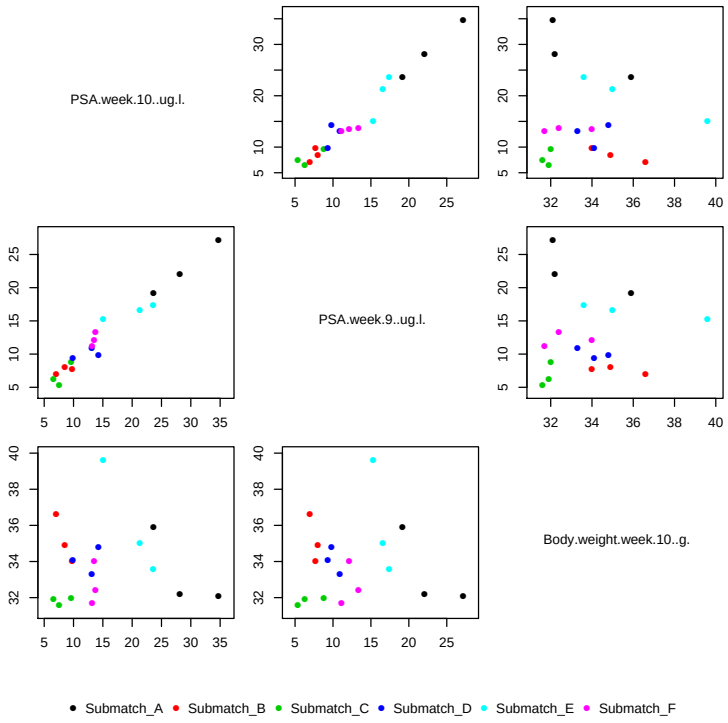

Figure 5: Test mixplot with submatch labels

```
> mixplot(ex[,c(2:4,6)], pch=16)
```

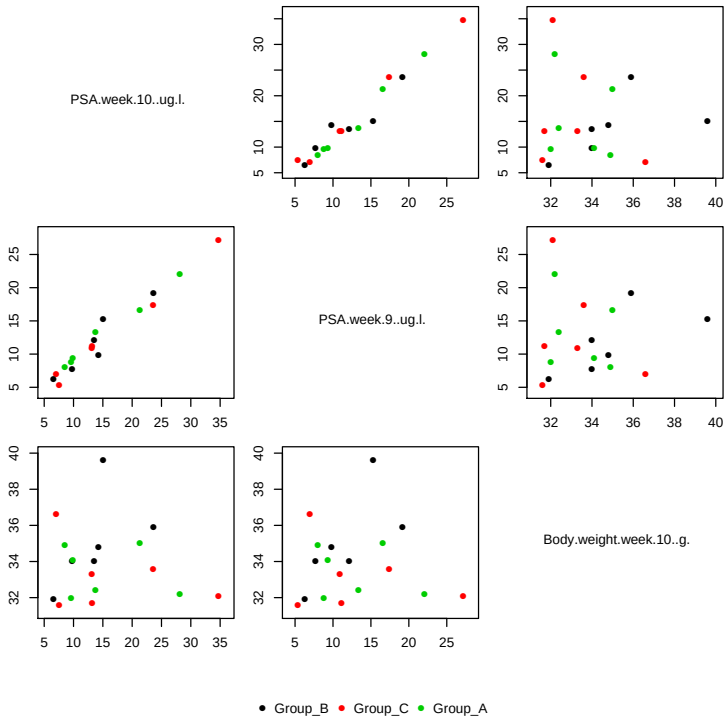

Figure 6: Test mixplot with allocation group labels

3 Power analysis

The power simulations provided by the `hamlet`-package are conducted through bootstrap (sampling with replacement) simulations using a pre-fitted mixed-effects model. For this purpose, it is essential that the user pre-defines a suitable mixed-effects model in the `lmer`-function of the `lme4`-package, as this will be used in the sampling process. The function `mem.powersimu` is the main `hamlet` function that performs this sampling, and it automatically identifies the suitable experimental unit from the `lme4`-object, and then re-fits the model structure a pre-defined amount of times at given N values.

3.1 An artificial example

In a situation where the user wishes to generate artificial data, it is important for the experimenter to evaluate such factors as:

- How many measurement points time will be available
- What is the expected effect size
- Will right-censoring (death or sacrifice) occur and what are the risk criteria
- Are there baseline differences or is there a correlation between the initial baselinse response level and intervention efficacy

The user is encouraged to creatively produce such expert curated data either by hand, or through a tailored simulation function. As a practical example, an example function will be constructed below (mainly utilizing normal distributions). In order for the artificial data to be modeled using `lme4`-package, it should follow the long format.

As an example, data with an initial baseline level of response values with $\mu = 5$ and $\sigma = 2$ will be generated from the normal distribution. 4 follow-up time points will be available after the initial baseline, and the expected control growth will be 2 per time point and in turn the intervention effect to have an effect of -1 to growth per time point. Furthermore, we simulate a right-censoring that has 20% chance to occur for individuals reaching above response values > 10. In this artificial example, 5 individuals will be available for both the control and the intervention group. Each measurement will have measurement error with no bias ($\mu = 0$) and $\sigma = 2$.

```
> # Baseline characteristics and time follow-up
> basemu <- 5
> basesigma <- 2
> ttime <- 4
> # Growth characteristics and group size
> growth <- 2
> interv <- -1
> ngroup <- 5
> # Measurement error and right-censoring
> measerror <- 2
> censthreshold <- 10
> censchance <- 0.2
```

```
> # Artificial data simulation with a set seed
> set.seed(1)
> # 2 experiment groups
> artdat <- do.call("rbind", lapply(c("Control", "Intervention"), FUN=function(group){
+  # Simulated individuals
+  do.call("rbind", lapply(1:ngroup, FUN=function(i){
+   # Baseline time = 0 and 5 follow up points
+   y <- rnorm(n = 1, mean=basemu, sd = basesigma)
+   # Growth as a function of time, with a possible intervention effect
+   measurements <- unlist(lapply(0:ttime, FUN=function(t){
+    y + growth*t + ifelse(group=="Intervention", interv*t, 0)
+   }))
+   # Random chance of censoring for response above >10,
+         # 20\% chance per time point to right-censor
+   for(index in 1:length(measurements)){
+    if(!is.na(measurements[index]) & measurements[index]>censthreshold)
+    if(rbinom(n=1, size=1, prob=censchance))
+          measurements[index:(length(measurements))] <- NA
+   }
+   # Add random measurement error
+   measurements <- measurements +
+    rnorm(n=length(measurements), mean=0, sd=measerror)
+         # Collect all data to a long format data.frame
+   data.frame(
+          Response = measurements,
+          ID = paste(group, i, sep="_"),
+          Group = ifelse(group=="Intervention", 1, 0),
+          Time = 0:ttime)
+  }))
+ }))
```

The above generated simulation script captures some key elements in a preclinical longitudinal intervention study, but should be naturally refined more precisely if more complex interactions are to be incorporated. To give insight into the overall structure of the long-format data, here are the so-called `head` and `tail` of the artificially generated `data.frame`:

```
> head(artdat)

  Response        ID Group Time
1 6.406691 Control_1     0    0
2 8.291951 Control_1     0    1
3 8.576375 Control_1     0    2
4 6.667192 Control_1     0    3
5 9.889958 Control_1     0    4
6 9.219866 Control_2     0    0

> tail(artdat)

   Response             ID Group Time
45 7.593970 Intervention_4     1    4
```

```
46 6.856897 Intervention_5     1    0
47 7.117348 Intervention_5     1    1
48 6.258141 Intervention_5     1    2
49 8.907789 Intervention_5     1    3
50 8.509502 Intervention_5     1    4
```

After a suitable long-format data.frame has been generated (variable `artdat` here), one has to specify and fit a preliminary mixed-effects model that will be used as a base for power simulations. The generated artificial data is presented in Figure 7.

3.1.1 Structure of a mixed-effects model

A standard mixed-effects model would, for example, include the following coefficients, given the input data `artdat` (the formula coefficients need to corresponds to column names in the input `data.frame`):

```
> f1a <- as.formula(Response ~ 1 + Time + Time:Group + (1 + Time|ID))
> f1b <- as.formula(Response ~ 1 + Time + Time:Group + (1|ID) + (0 + Time|ID))
```

This formula is structured as follows:

- The left hand side `Response` is our response vector `y`.
- The non-parenthesis coefficients following the tilde are the so called *fixed effects*, which are here population-wise parameters
- The first right hand side coefficient `1` stands for standard model intercept, i.e. y level when $x = 0$
- Coefficient `Time` captures natural growth of the tumors as a function of time
- Coefficient `Time:Group` introduces grouping information as an interaction with the growth coefficient, and thus tests whether the intervention gives a growth inhibition advantage.
- The terms in parenthesis are *random effects* with analogous counterparts to their *fixed effects*. The difference is that the grouping variable, indicated here with `|ID`, is gives flexibility for the each experimental unit to have deviating intercepts and growth slopes. Separate value from a normal distribution with mean 0 and an estimated standard deviation are identified when fitting the mixed-effects model. The *random effects* allow individualized response curves, while controlling that multiple observations belong to a single individual (`ID`).
- An alternate non-correlated random-effects structure is given in `f1b`, indicated by separating the two *random effects* terms without a cross-correlation.

Fixed effects are typically utilized in inference of possible intervention effects, and here the term `Time:Group` will estimate possible intervention effects. A linear mixed-effects model of the above structure can be fitted using:

```
> # Plot the artificial data
> plot.new()
> plot.window(xlim=range(artdat[,"Time"]),
+         ylim=c(0,max(artdat[,"Response"], na.rm=T)))
> axis(1); axis(2); box()
> title(xlab="t", ylab="y", main="Artificially generated data")
> # Plot each individual as its own curve
> invisible(by(artdat, INDICES=artdat[,"ID"], FUN=function(z){
+         points(z[,"Time"], z[,"Response"], type="l", col=1+z[1,"Group"])
+         points(z[,"Time"], z[,"Response"], pch=16, col=1+z[1,"Group"])
+ }))
> legend("bottomright", col=1:2, pch=16, lwd=1, legend=c("Control", "Intervention"))
```

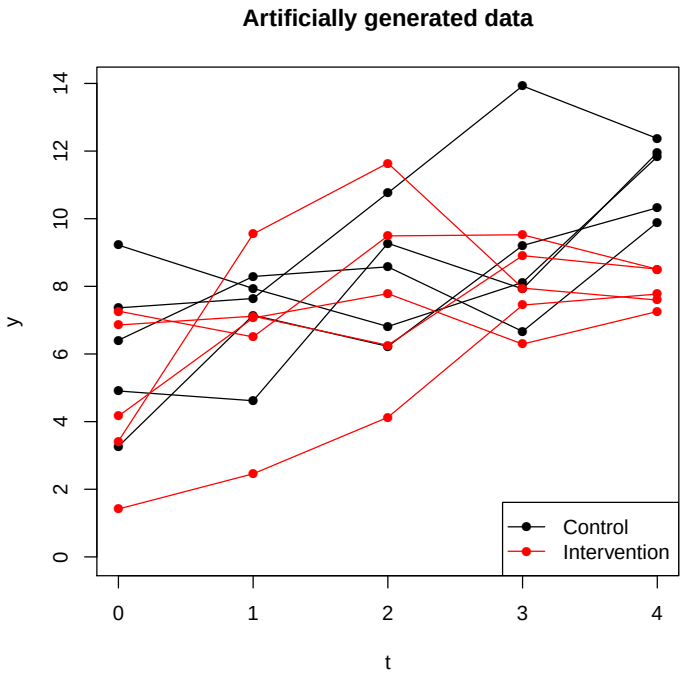

Figure 7: Visualization of the artificially generated data, with two experimental groups.

```
> library(lme4)
> # We defined formulae already before
> fit1 <- lmer(f1b, data = artdat)
> library(lmerTest)
> summary(fit1)

Linear mixed model fit by REML ['lmerMod']
Formula: Response ~ 1 + Time + Time:Group + (1 | ID) + (0 + Time | ID)
   Data: artdat

REML criterion at convergence: 204.8

Scaled residuals:
    Min      1Q  Median      3Q     Max
-1.6891 -0.6997  0.1232  0.5499  2.3074

Random effects:
 Groups   Name        Variance Std.Dev.
 ID       (Intercept) 1.207    1.099
 ID.1     Time        0.000    0.000
 Residual             2.830    1.682
Number of obs: 50, groups:  ID, 10

Fixed effects:
            Estimate Std. Error t value
(Intercept)   5.6894     0.5390  10.556
Time          1.2732     0.2135   5.964
Time:Group   -0.5251     0.2629  -1.998

Correlation of Fixed Effects:
           (Intr) Time
Time       -0.492
Time:Group  0.000 -0.616
```

3.1.2 Bootstrap simulations

The package `lmerTest` is used to provide Satterthwaite approximation for the p-values for fixed effects in the linear mixed-effects model. Albeit the p-values are provided here for the model coefficients, we are interested in how power in such a study would develop as a function of animal numbers N. For this purpose we can perform power simulations, which bootstraps the pre-fitted mixed-effects model on our artificial data:

Notice that the artificial data simulations were run with a very limited bootstrap sample size, in order to save time in generation of this vignette. A better estimate for the power as well as more exact N would be given e.g. by setting `boot=1000` and `N=3:20`. Furthermore, we indicated with `level` that our experimental unit is defined by the individual indicator `ID`, and that we want to subsample evenly over the intervention groups through `strata`. The resulting power curve is shown in Figure 8, suggesting that in order to achieve sufficient statistical power, our experiment should include at least $N_{arm} = 9$ individuals

```
> set.seed(1)
> pow <- mem.powersimu(fit1,
+         N=c(3, 5, 7, 9, 11, 13, 15), boot=20,
+         level="ID", strata="Group")
> abline(h=0.8, col="grey")
> pow
```

	(Intercept)	Time	Time:Group
GroupN_3_TotalN_6	1	1	0.35
GroupN_5_TotalN_10	1	1	0.25
GroupN_7_TotalN_14	1	1	0.70
GroupN_9_TotalN_18	1	1	0.80
GroupN_11_TotalN_22	1	1	0.80
GroupN_13_TotalN_26	1	1	1.00
GroupN_15_TotalN_30	1	1	0.95

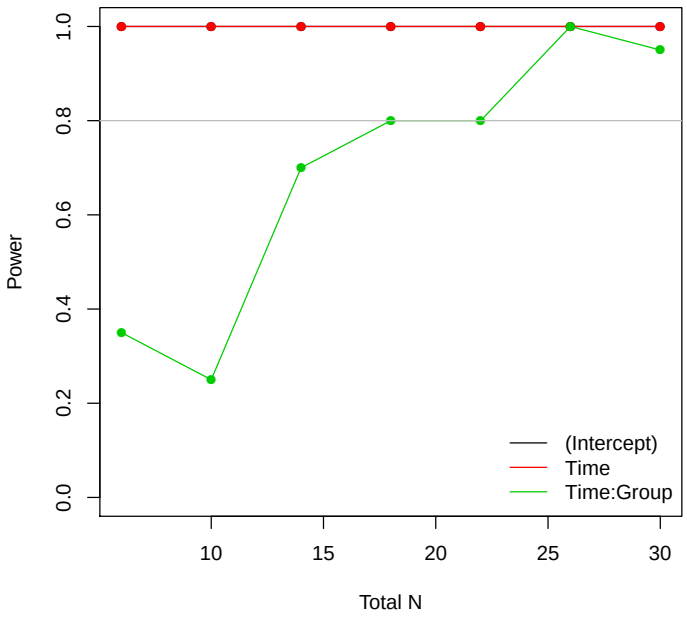

Figure 8: Preliminary power curve for all the fixed effects, with a limited number of bootstrapped datasets (20).

in both intervention arms, and total experiment size consisting of $N_{total} = 18$ individuals.

3.2 An ARN-509 example

The longitudinal intervention observations of the ARN-509 / MDV3100 -study and ORX / ORX+Tx -study are provided inside the `hamlet`-package with the commands `data(vcaplong)` and `data(orxlong)`, respectively. Here we will provide a model fit to the ARN-509 -study, as well as show how its power curve behaves in respect to different *fixed effects*. Load the ARN-509 / MDV3100 -study and constraint to ARN-509 by:

```
> data(vcaplong)
> arndat <- vcaplong[
+         # Select observations only from the vehicle or ARN-509 groups
+         vcaplong[,"Group"] %in% c("Vehicle", "ARN"),
+         # Select columns (=features) that are required for the conventional MEM
+         c("PSA", "DrugWeek", "ARN", "ID")]
> head(arndat)

     PSA DrugWeek ARN    ID
11 21.30        0   0 ID003
12 45.69        1   0 ID003
13 54.50        2   0 ID003
14 53.55        3   0 ID003
15 27.64        4   0 ID003
71 13.17        0   0 ID009
```

Similarly as for the artificial data example, this study could be be representative for estimating power for interventions with similar effect sizes, censoring, follow-up periods etc. The conventional non-matched modeling process and corresponding preliminary power curve would be computed using:

```
> arnfit <- lmer(PSA ~ 1 + DrugWeek + DrugWeek:ARN + (1|ID) + (0 + DrugWeek|ID),
+         data = arndat)
> summary(arnfit)

Linear mixed model fit by REML t-tests use Satterthwaite approximations to degrees
  of freedom [lmerMod]
Formula: PSA ~ 1 + DrugWeek + DrugWeek:ARN + (1 | ID) + (0 + DrugWeek |      ID)
   Data: arndat

REML criterion at convergence: 1082.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.4768 -0.3911 -0.0044  0.3425  3.1437

Random effects:
 Groups   Name        Variance Std.Dev.
 ID       (Intercept) 67.80    8.234
```

```
 ID.1     DrugWeek    26.65    5.163
 Residual             33.05    5.749
Number of obs: 150, groups:  ID, 30

Fixed effects:
             Estimate Std. Error     df t value Pr(>|t|)
(Intercept)    14.311      1.709 29.587   8.374 2.68e-09 ***
DrugWeek       10.062      1.407 28.206   7.150 8.47e-08 ***
DrugWeek:ARN   -7.627      1.982 27.792  -3.849 0.000636 ***
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Correlation of Fixed Effects:
            (Intr) DrugWk
DrugWeek    -0.092
DrugWek:ARN  0.000 -0.704
```

As is seen in Figure 9, the power curves become smoother with higher bootstrap rates. Here a highly narrowed N vector was tested (values 5 to 9), due to a priori knowledge that the power 0.8 would be achieved at $N = 7$.

```
> set.seed(123)
> arnpow <- mem.powersimu(arnfit,
+          level = "ID", strata = "ARN",
+          N = c(5,6,7,8,9), boot = 100)
> abline(h=0.8, col="grey")
> arnpow
```

	(Intercept)	DrugWeek	DrugWeek:ARN
GroupN_5_TotalN_10	1	0.98	0.68
GroupN_6_TotalN_12	1	1.00	0.75
GroupN_7_TotalN_14	1	1.00	0.90
GroupN_8_TotalN_16	1	1.00	0.83
GroupN_9_TotalN_18	1	1.00	0.93

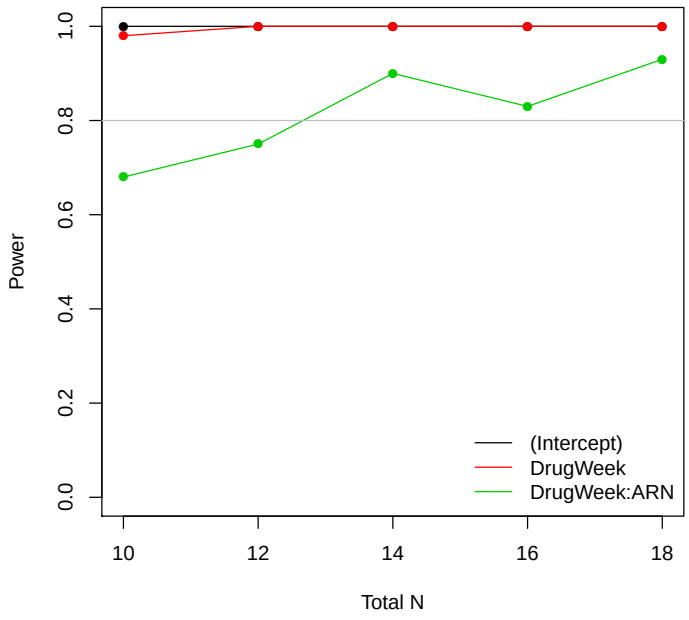

Figure 9: ARN-509 fixed effects power curves, estimated using the conventional model with 100 bootstrapped data sets.

4 Post-intervention analyses

The presented pre-intervention submatching procedure provides a unique opportunity to utilize this predictive power to improve accuracy in the post-intervention inference. We here provide the datasets from both the *ARN-509 / MDV3100* -study and the *ORX / ORX+Tx* -study, by typing `data(vcaplong)` and `data(orxlong)`, respectively. Alternatively, the pre-intervention data is also available in `data(vcapwide)` and `data(orxwide)`, respectively.

4.1 Long format and the presented datasets

As out-lined before, in order to perform regression modeling, R requires the observations to be in the so-called long format, where each row in a data.frame corresponds to a single measurement. These measurements are then usually uniquely defined using individual identification codes as well as time points. The presented datasets:

```
> data(vcaplong)
> data(orxlong)
> head(vcaplong)
```

```
     PSA  log2PSA   BW    Submatch    ID Week DrugWeek   Group Vehicle ARN MDV
11 21.30 4.412782 35.0  Submatch_1 ID003   10        0 Vehicle       1   0   0
12 45.69 5.513807 36.1  Submatch_1 ID003   11        1 Vehicle       1   0   0
13 54.50 5.768184 37.9  Submatch_1 ID003   12        2 Vehicle       1   0   0
14 53.55 5.742815 37.5  Submatch_1 ID003   13        3 Vehicle       1   0   0
15 27.64 4.788686 39.7  Submatch_1 ID003   14        4 Vehicle       1   0   0
41  7.55 2.916477 31.6 Submatch_10 ID007   10        0     MDV       0   0   1
```

```
> head(orxlong)
```

```
   ID   PSA    log2PSA Day TrDay       Date  Group    Submatch ORXTx ORX Intact
1 ID1 0.368 -1.4422223   0   -10 2015-01-12 ORX+Tx Submatch_11     1   0      0
2 ID1 1.524  0.6078629  10     0 2015-01-22 ORX+Tx Submatch_11     1   0      0
3 ID1 0.034 -4.8783214  25    15 2015-02-06 ORX+Tx Submatch_11     1   0      0
4 ID1 0.100 -3.3219281  35    25 2015-02-16 ORX+Tx Submatch_11     1   0      0
5 ID1 0.203 -2.3004484  45    35 2015-02-26 ORX+Tx Submatch_11     1   0      0
6 ID1 0.357 -1.4860040  56    46 2015-03-09 ORX+Tx Submatch_11     1   0      0
```

Typically, an experimenter may model a single interesting contrast with a single model, thus we will split the `vcaplong` and `orxlong` into two separate data sets.

```
> # Interesting fields in the orthotopic VCaP study
> fields <- c('PSA', 'DrugWeek', 'ID', 'Submatch', 'Group', 'ARN', 'MDV')
> # ARN-509 vs Vehicle
> arndat <- vcaplong[vcaplong[,'Group'] %in% c('ARN', 'Vehicle'),fields]
> # MDV3100 vs Vehicle
> mdvdat <- vcaplong[vcaplong[,'Group'] %in% c('MDV', 'Vehicle'),fields]
> # Interesting fields in the subcutaneous VCaP study
> fields <- c('PSA', 'TrDay', 'ID', 'Submatch', 'Group', 'ORXTx', 'ORX')
```

```
> # ORX vs Intact
> orxdat <- orxlong[orxlong,'Group'] %in% c('ORX', 'Intact'), fields]
> # ORX+Tx vs ORX
> xtxdat <- orxlong[orxlong,'Group'] %in% c('ORX+Tx', 'ORX'), fields]
```

4.2 Collating to pairwise submatched observations

In order to fit pairwise matched mixed-effects models, the experimenter should utilize the baseline submatch information to subtract corresponding control growth from its intervened counterpart. For this, a field indicating *Submatch* should be available in the data frame, and subtraction in a pairwise manner per each time point $Time$. For example:

```
> arndat <- arndat[order(arndat,'Submatch']),]
> arnpair <- do.call('rbind', by(arndat, INDICES=arndat,'Submatch'], FUN=function(z){
+         # Within each Submatch, subtract Vehicle from the Case
+         z,'PairPSA'] <- z,'PSA'] - z[z,'Group']=='Vehicle','PSA']
+         z
+ }))
> arnpair[1:10,]
```

	PSA	DrugWeek	ID	Submatch	Group	ARN	MDV	PairPSA
Submatch_1.11	21.30	0	ID003	Submatch_1	Vehicle	0	0	0.00
Submatch_1.12	45.69	1	ID003	Submatch_1	Vehicle	0	0	0.00
Submatch_1.13	54.50	2	ID003	Submatch_1	Vehicle	0	0	0.00
Submatch_1.14	53.55	3	ID003	Submatch_1	Vehicle	0	0	0.00
Submatch_1.15	27.64	4	ID003	Submatch_1	Vehicle	0	0	0.00
Submatch_1.266	23.62	0	ID040	Submatch_1	ARN	1	0	2.32
Submatch_1.267	22.09	1	ID040	Submatch_1	ARN	1	0	-23.60
Submatch_1.268	30.95	2	ID040	Submatch_1	ARN	1	0	-23.55
Submatch_1.269	31.98	3	ID040	Submatch_1	ARN	1	0	-21.57
Submatch_1.270	41.54	4	ID040	Submatch_1	ARN	1	0	13.90

```
> # The vehicle observations are redundant (subtracted from themselves)
> arnpair <- arnpair[arnpair,'Group']=='ARN',]
```

In the above example, first pairwise computed PSA results in $23.62 - 21.30 = 2.32$ at time point $DrugWeek = 0$ (baseline). The following time points, i.e. $DrugWeek = 1$ result in turn a much more drastic growth in control tumor, i.e. $22.09 - 45.69 = -23.60$. In this particular example, the treated tumor seems to grow much slower for 3 weeks subsequently to the baseline, until it bounces back in the final time point.

4.3 Fitting conventional and pairwise matched mixed-effects models

Linear mixed-effects models compose of two main components:

- **Fixed effects**; Population effects, usually considered to cover either whole range of experimental units or a subpopulation such as an intervention group

- **Random effects**; Individual effects, that allows flexible model fits. Typical random effects include a random intercept (variation at baseline) and a random slope (individual variation in the growth coefficient).

The formula interface in R for fitting linear mixed-effects models in `lme4`-package consists of three parts:

$$LFS \sim \mathbf{Xb} + \mathbf{Zu} + \epsilon \tag{1}$$

where the LFS refers to left-hand side, i.e. the response vector $\mathbf{y}$ which is usually a tumor growth feature such as serum PSA or tumor volume. The right hand side from $\sim$ holds the fixed effects part (here $\mathbf{Xb}$), random effects part (here $\mathbf{Zu}$) and the normally distributed error term ϵ.

Fixed effects are separated using the + sign. A typical longitudinal preclinical model could be built on three fixed effects terms:

$$1 + Time + Time : Group \tag{2}$$

where 1 refers to a common intercept, which could be alternatively omitted using either a 0 or a -1 sign instead of 1. $Time$ typically is a running time point indicator, that starts from 0 at baseline and ranges to certain time units such as weeks or days, and tumor growth is computed a slope coefficient as a function of time. Furthermore, the term $Time : Group$ adds a binary indicator $Group$ that may be used to compare a subpopulation in comparison to control tumor growth.

Furthermore, random effects follow a notation:

$$\begin{cases} +(1 + Time|GroupingFactor) \\ +(1|GroupingFactor) + (0 + Time|GroupingFactor) \end{cases} \tag{3}$$

where the notation indicates that each unique factor value within variable `GroupingFactor` is treated as an instance of the experimental unit. The upper notation indicates that both an individual-specific intercept as well as an individualized time-dependent slope are estimated along with a cross-covariance between the two normally distributed random effects. The lower notation in turn estimates these two effects, an individual-specific intercept and an individualized time-dependent slope, separately from each other. By default we utilized the lower notation approach, though the upper notation may offer an interesting alternative. The error term does not need to explicitly included in the model formula. In the ARN-509 -study, the presented mixed-effects models were fitted using:

```
> fit_arn_unmatched <- lmer(PSA ~ 1 + DrugWeek + DrugWeek:ARN
+         + (1|ID) + (0 + DrugWeek|ID), data = arndat)
> fit_arn_matched <- lmer(PairPSA ~ 0 + DrugWeek
+         + (1|ID) + (0 + DrugWeek|ID), data = arnpair)
> summary(fit_arn_unmatched)

Linear mixed model fit by REML t-tests use Satterthwaite approximations to degrees
  of freedom [lmerMod]
Formula: PSA ~ 1 + DrugWeek + DrugWeek:ARN + (1 | ID) + (0 + DrugWeek |      ID)
   Data: arndat
```

```
REML criterion at convergence: 1082.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.4768 -0.3911 -0.0044  0.3425  3.1437

Random effects:
 Groups   Name        Variance Std.Dev.
 ID       (Intercept) 67.80    8.234
 ID.1     DrugWeek    26.65    5.163
 Residual             33.05    5.749
Number of obs: 150, groups:  ID, 30

Fixed effects:
             Estimate Std. Error     df t value Pr(>|t|)
(Intercept)    14.311      1.709 29.587   8.374 2.68e-09 ***
DrugWeek       10.062      1.407 28.206   7.150 8.47e-08 ***
DrugWeek:ARN   -7.627      1.982 27.792  -3.849 0.000636 ***
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Correlation of Fixed Effects:
            (Intr) DrugWk
DrugWeek    -0.092
DrugWek:ARN  0.000 -0.704
```

> summary(fit_arn_matched)

```
Linear mixed model fit by REML t-tests use Satterthwaite approximations to degrees
  of freedom [lmerMod]
Formula: PairPSA ~ 0 + DrugWeek + (1 | ID) + (0 + DrugWeek | ID)
   Data: arnpair

REML criterion at convergence: 592.8

Scaled residuals:
     Min       1Q   Median       3Q      Max
-2.36132 -0.53405 -0.04075  0.34125  2.72586

Random effects:
 Groups   Name        Variance Std.Dev.
 ID       (Intercept) 49.74    7.053
 ID.1     DrugWeek    79.11    8.894
 Residual             70.55    8.399
Number of obs: 75, groups:  ID, 15

Fixed effects:
         Estimate Std. Error     df t value Pr(>|t|)
DrugWeek   -7.962      2.366 13.770  -3.365  0.00472 **
```

```
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1
```

Alternate pairwise-matched longitudinal model formulations could include for example:

```
> # Paired model with correlated random effects
> fit_arn_matched2 <- lmer(PairPSA ~ 0 + DrugWeek
+         + (1 + DrugWeek|ID), data = arnpair)
> # Paired model with an intercept fixed effect
> fit_arn_matched3 <- lmer(PairPSA ~ 1 + DrugWeek
+         + (1|ID) + (0 + DrugWeek|ID), data = arnpair)
> # Paired model with an intercept fixed effect
> # and correlated random effects
> fit_arn_matched4 <- lmer(PairPSA ~ 1 + DrugWeek
+         + (1 + DrugWeek|ID), data = arnpair)
> # Only fixed effects shown here to save space
> summary(fit_arn_matched2)$coefficients

          Estimate Std. Error df   t value   Pr(>|t|)
DrugWeek -5.840363   2.122596 14 -2.751519 0.01559838

> summary(fit_arn_matched3)$coefficients

             Estimate Std. Error       df   t value    Pr(>|t|)
(Intercept) -4.305333   2.295298 14.28259 -1.875719 0.081283732
DrugWeek    -7.302533   2.423880 14.28259 -3.012745 0.009142255

> summary(fit_arn_matched4)$coefficients

             Estimate Std. Error       df   t value    Pr(>|t|)
(Intercept) -4.305333   2.122769 13.99982 -2.028169 0.062013340
DrugWeek    -7.302533   2.241686 13.99999 -3.257608 0.005725389
```

For fitted models, further functions are provided inside `hamlet`, both for visualization as well as diagnostics purposes. Examples:

- `hamlet::mem.plotresid` for plotting residuals along with trend lines
- `hamlet::mem.getcomp` for extracting a `data.frame` containing observation-specific fixed effects fit, full model fit, response vector etc. These can be then used to visualize the corresponding model fits, for example by paneling for experimental groups or each individual participating in the study.